MARRI LAXMAN REDDY GROUP OF INSTITUTIONS

MLR INSTITUTE OF TECHNOLOGY

(UGC AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE

Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India

A
Mini Project Report

On

"WildDiscover: AI-Powered Wildlife & Landmark Explorer"

Submitted in partial fulfillment of the
Requirements for the award of the degree of

Bachelor of Technology

In

Computer Science & Engineering-

Artificial Intelligence & Machine Learning

By

K. Harsha – 22R21A6695

Under the guidance of

Mrs. G. UMAMAHESWARI
Assistant Professor

Department of Computer Science & Engineering-
Artificial Intelligence & Machine Learning

2022-2026

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS
MLR INSTITUTE OF TECHNOLOGY
(UGC AUTONOMOUS)
Affiliated to JNTUH, Approved by AICTE
Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India

Department of Computer Science & Engineering-Artificial Intelligence & Machine Learning

CERTIFICATE

This is to certify that the project entitled **"WildDiscover: AI-Powered Wildlife & Landmark Explorer"** has been submitted by **K. Harsha (22R21A6695)**in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering-Artificial Intelligence & Machine Learning from Jawaharlal Nehru Technological University, Hyderabad. The results embodied in this project have not been submitted to any other University or Institution for the award of any degree or diploma.

Internal Guide **Project-Coordinator**

Head of the Department **External Examiner**

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS
MLR INSTITUTE OF TECHNOLOGY
(UGC AUTONOMOUS)
Affiliated to JNTUH, Approved by AICTE
Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India
NBA NATIONAL BOARD OF ACCREDITATION
NAAC

Department of Computer Science & Engineering- Artificial Intelligence & Machine Learning

DECLARATION

We hereby declare that the project entitled **"WildDiscover: AI-Powered Wildlife & Landmark Explorer"** is the work done during the period from **February 2025 to June 2025** and is submitted in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in **Computer Science and Engineering- Artificial Intelligence & Machine Learning** from Jawaharlal Nehru Technology University, Hyderabad. The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

K. Harsha **22R21A6695**

Department of Computer Science & Engineering- Artificial Intelligence & Machine Learning

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that we now have the opportunity to express our guidance for all of them.

First of all, we would like to express our deep gratitude towards our internal guide **G.UMAMAHESWARI, Assistant Professor, Department of CSE- Artificial Intelligence & Machine Learning** for his support in the completion of our dissertation. We wish to express our sincere thanks to **Mr. K. Sai Prasad, HOD, Department of CSE- Artificial Intelligence & Machine Learning for providing the facilities to complete the dissertation.**

We would like to thank all our faculty and friends for their help and constructive criticism during the project period. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

K. Harsha **22R21A6695**

Department of Computer Science & Engineering-

Artificial Intelligence & Machine Learning

ABSTRACT

Mobile applications for environmental awareness and exploration have become increasingly popular in recent years. People are becoming more interested in recognizing landmarks, wildlife, and plants when they are outdoors, but they frequently lack trustworthy tools for in-the-moment identification. Conventional apps have limited offline support and rely significantly on internet connectivity, which can be problematic in remote locations. In order to get around this, we created WildDiscover, a React Native application that uses on-device machine learning in conjunction with cloud-based APIs to provide precise image recognition even when offline. Locally, the app classifies plants and animals using lightweight MobileNet models. For improved accuracy online, it integrates with services like Google Vision and PlantNet. Wikipedia integration enhances the educational process, and identified items are saved on AWS S3 with metadata for later use. Reliability, speed, and educational value in nature and landmark exploration are guaranteed by this hybrid approach.

LIST OF FIGURES & TABLES

INDEX

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

People frequently take pictures of plants, animals, or landmarks without knowing their names or other details in order to use mobile applications to explore and learn about the world around them. Many have trouble identifying species in remote locations or lack trustworthy offline tools. To meet this need, WildDiscover provides a clever, intuitive mobile application that uses a combination of cloud-based APIs and on-device machine learning to identify plants, animals, and landmarks. The application incorporates Wikipedia for comprehensive information, AWS S3 for safe image storage, and MobileNet for offline predictions. Even without internet access, users can view identification results instantly. The system uses highly accurate image recognition models, which improves education and raises awareness of environmental issues.For inquisitive users, hikers, tourists, and researchers, WildDiscover provides a robust and dynamic platform that allows them to instantly identify natural elements and save them for later use.

1.2 PURPOSE OF THE PROJECT

The goal of this project is to develop a mobile application that uses image recognition and machine learning techniques to identify landmarks, plants, and animals. The application seeks to give users an easy-to-use platform so they can explore and discover their surroundings in real time. This application allows users to quickly identify unknown species or locations in photos they take. In order to ensure accessibility even in remote locations, the app incorporates a lightweight machine learning model (MobileNet) for offline use.

1.3 MOTIVATION

Researchers and outdoor enthusiasts are increasingly using mobile applications for nature identification and exploration. These apps' ease of use and accessibility have ushered in a new era of environmental education. Users take innumerable pictures of landmarks, animals, and plants every day, producing a useful data source. Unfortunately, a lot of users have trouble correctly recognizing these natural elements, which hinders their ability to learn. We were motivated by this challenge to create WildDiscover, a cutting-edge application that makes use of machine learning and image recognition technologies. By enabling users to quickly and accurately identify plants, animals, and landmarks, WildDiscover hopes to strengthen users' ties to the natural world and raise awareness of environmental issues.

CHAPTER 2

LITERATURE SURVEY

By examining current mobile applications and machine learning models for plant, animal, and landmark identification, we carried out a thorough literature review. In order to obtain knowledge and best practices for creating a precise and efficient identification system, research papers, journals, and publications were also examined.

2.1 EXISTING SYSTEM

1. **Neupane (2022)**

   This paper compares computer vision techniques in wildlife monitoring using deep learning, IoT, and camera trap data. It highlights improvements in accuracy and object tracking. However, the study identifies a gap in image preprocessing techniques.

2. **Mpouziotas (2023)**

   The authors developed a bird detection system from drone footage using the YOLOv4 model. It achieved high precision (91.28%) in real-world detection. Lack of dataset size details affected model generalizability.

3. **Sazida Binta Islam (2023)**

   This study used deep CNNs to classify species from ecological camera trap images. It demonstrated efficient species recognition on large datasets. More research is needed to handle rare species and varied environments.

4. **J. Abdollahi (2022)**

   The paper explores deep learning for identifying medicinal plant species through image processing. It supports conservation efforts using AI-based classification. Dataset details were not provided, limiting reproducibility.

5. **Andres Hernandez (2024)**

   Pytorch-Wildlife is introduced as a collaborative framework for AI-based wildlife conservation. It achieved high accuracy in identifying invasive and Amazonian species. However, dataset diversity and scalability remain challenges.

6. **ME Hossain (2022)**

   This review outlines deep learning models in wildlife conservation and monitoring. It highlights success in accuracy and practical deployment. Still, issues with data availability and model interpretability persist.

7. **C. Jo (2022)**

   A CNN-based model was proposed for anatomical landmark detection in full-leg radiographs. It showed accurate angle measurements for clinical use. Broader validation across populations is recommended.

8. **Razali (2023)**

   The paper presents a CNN + LDA-based lightweight landmark recognition model for smart tourism. It achieved high classification accuracy on specific datasets. The model's general applicability across various scenes is yet to be confirmed.

9. **J. Deka (2023)**

   This study automates freshwater fish species classification using deep CNNs like ResNet-50. ResNet-50 outperformed AlexNet in terms of accuracy and efficiency. Larger and more diverse datasets are needed.

10. **Sukanta Ghosh (2022)**

    The paper proposes hybrid CNN, SVM, and KNN architectures for plant leaf classification. It achieved 99.5% accuracy across selected datasets. However, further testing on broader samples is needed.

11. Kosuke Takaya (2024)

This research used EfficientNetV2 and YOLOv5 for salamander identification via smartphone images. The method achieved near-perfect accuracy. Field testing and long-term application are still required.

12. Carvajal (2022)

A review of deep learning approaches in wildlife detection using camera trap images was conducted. It confirmed improved monitoring accuracy. The paper highlights the need for more diverse and robust datasets.

2.2 LIMITATIONS OF EXISTING SYSTEM

Concisely summarizing the disadvantages of the above implementations:

- The study emphasizes computer vision techniques but gives minimal attention to advanced image preprocessing, which is essential for improving detection performance from noisy camera trap images.

- The absence of clear dataset size and diversity reduces confidence in the model's reproducibility and weakens its potential to generalize across different species or habitats.

- The proposed system performs well in controlled conditions but struggles to identify rare or visually similar species, especially when applied to diverse or unstructured environments.

- Lack of transparency in dataset structure and characteristics hinders reproducibility, benchmarking, and practical adaptation for real-world deployment.
- Although the model achieved high accuracy, it was trained on a limited dataset lacking geographic and species diversity, which restricts its application to global conservation efforts.

- There's a noticeable gap in standardization and inter-institutional collaboration, resulting in fragmented research progress and difficulties in developing scalable wildlife conservation tools.

- The AI system was tested only on specific demographics and image types, requiring broader validation across varied imaging devices, populations, and scenarios for true generalization.

- The model's high performance was restricted to a limited landmark dataset and does not guarantee similar results in complex or unfamiliar landmark environments across different regions.

- Classification was done using a narrow set of fish species, which makes the model unsuitable for deployment in environments with a broader variety of aquatic life.

- The plant classification models were evaluated only on select datasets and may fail to perform in field conditions where lighting, angles, and occlusion vary significantly.

- The solution shows excellent accuracy in lab conditions but lacks real-world field testing, which is crucial to assess its reliability for long-term monitoring in dynamic environments.

- Despite proving the effectiveness of deep learning in wildlife monitoring, the paper points out that most available datasets lack species diversity, temporal depth, and regional coverage, limiting robustness.

CHAPTER 3

PROPOSED SYSTEM

3.1 PROPOSED SYSTEM

Creating a mobile application that allows users to take or upload pictures of plants, animals, or landmarks they encounter while exploring is the suggested strategy. A machine learning classification model built on the MobileNet architecture is used to identify the image. A carefully chosen dataset of labeled photos is gathered in order to train the classification model, and different data preprocessing and augmentation methods are used to increase accuracy and resilience in practical situations.

3.2 OBJECTIVES OF PROPOSED SYSTEM

The following are among the goals of the suggested system:

- To use machine learning to recognize landmarks, plants, and animals.
- To use lightweight on-device models to offer offline accessibility.
- To gather and prepare a labeled image dataset for model training.
- To use MobileNet, a convolutional neural network model, for image classification.

3.3 SYSTEM REQUIREMENTS

Here are the requirements for developing and deploying the application.

3.3.1 SOFTWARE REQUIREMENTS

The following are the software specifications needed to develop the application:

1. Python and JavaScript are necessary (for React Native).
2. VSCode or Android Studio are code development editors.
3. ML libraries for model building, including TensorFlow and Keras
4. To test the application, use a mobile device or emulator that supports cameras.

3.3.2 HARDWARE REQUIREMENTS

The following are the hardware specifications needed to develop the application:

1. Operating System: Linux, macOS, or Windows

2. Processor: at least an Intel i3

3. RAM: at least 4 GB

4. Hard drive: at least 250 GB; for better performance, an SSD is advised.

3.3.3 FUNCTIONAL REQUIREMENTS

1. Users should be able to take or upload pictures of plants, animals, or landmarks for identification through the application's user interface.
2. Before being sent to the model, the image data should be preprocessed using methods like resizing, normalization, and augmentation.
3. The input image should be classified by the trained MobileNet model, which will then produce the relevant label as an output (such as a landmark or species name).
4. To guarantee dependability, the system should assess model performance using metrics like accuracy, precision, recall, and confusion matrix.

3.3.4 NON-FUNCTIONAL REQUIREMENTS

Reliability

- The system should be able to correctly identify the image content no matter how many attempts are made.
- Any exceptions or invalid inputs should be handled gracefully by the system.
- Regarding the output, the system ought to be able to deliver dependable and prompt identification results.

Scalability

- The system should be able to reliably differentiate between various plant, animal, and landmark categories while maintaining its usability in order to yield better results.
- The system needs to be flexible enough to incorporate new model updates in the future without degrading performance.

3.4 CONCEPTS USED IN THE PROPOSED SYSTEM

DATA PREPROCESSING

Preprocessing is a data mining technique used to turn the raw data into a format that is both practical and effective.

MACHINE LEARNING AND ITS TYPES

Machine learning (ML) is a branch of study devoted to comprehending and developing "learning" methods, i.e., methods that use data to enhance performance on a certain set of tasks.

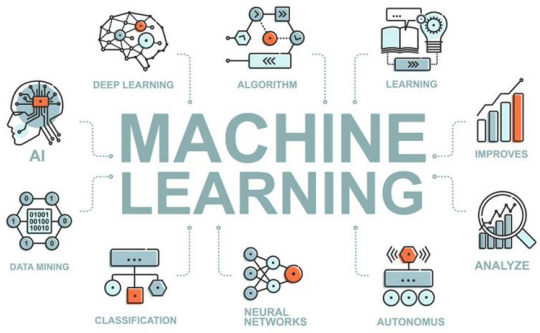

Figure 1: Types of Machine Learning

CLASSIFICATION

In machine learning and statistics, classification is a supervised learning technique in which the computer program makes new observations or classifications based on the data that is provided to it. It is a method of classifying a set of data into groups. It may be used with both structured and unstructured data. Predicting the class of the provided data points is the first step in the procedure. The terms target, label, and classes are frequently used to describe the classes.

SUPPORT VECTOR MACHINE

The SVM technique is used to develop the optimum decision boundary or line that can divide n-dimensional space into classes, allowing us to quickly classify additional data points in the future. A hyperplane is the name given to this optimal decision boundary.

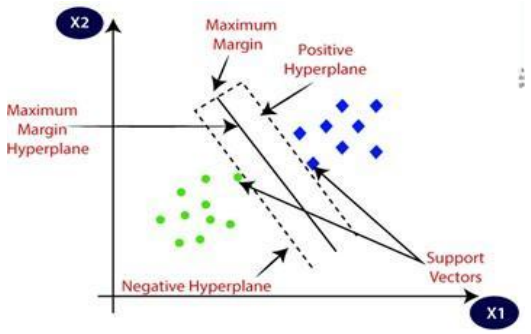

Figure 2: Representation of Support Vector Machine

3.4 DATA SET USED IN THE PROPOSED SYSTEM

Labeled photos of plants, animals, insects, and natural landmarks make up the varied dataset used by the WildDiscover app. These photos were gathered from openly accessible websites such as Wikimedia Commons, PlantCLEF, and iNaturalist. The dataset is improved using data augmentation techniques and contains an offline subset to facilitate identification without internet access, guaranteeing precise and effective landmark, flora, and fauna recognition.

CHAPTER 4

SYSTEM DESIGN

4.1 COMPONENTS OR USERS IN THE PROPOSED SYSTEM

- **Admin**

  The administrator is in charge of gathering and compiling a dataset of images with labels. The images are cleaned and standardized using feature extraction and data preprocessing techniques. To enable precise classification, the administrator then uses the MobileNet architecture to train the machine learning model.

- **ML Model/Classifier**

  The administrator uses the preprocessed dataset to train the MobileNet-based classifier. The application's built-in trained model is in charge of analyzing user-submitted photos in order to recognize and categorize them as belonging to a particular plant, animal, or landmark.

- **End user**

  The end user is any person who uses the WildDiscover mobile application to take or upload an image in order to identify a plant, animal, or landmark. Following submission, the image is processed by the model, which then provides the user with the relevant identification result.

4.2 PROPOSED SYSTEM ARCHITECTURE

A method for recognizing landmarks and natural features is shown in this diagram. Users can choose categories (Landmarks, Fauna, and Flora) and submit a photo using a "Discovery Screen" on the client side. Based on the category for identification, a server-side "API Gateway" coordinates the routing of this image to particular external APIs (such as PlantNet, Animal API, and Google Vision). After that, the "Result" is brought back to the client's "Result Screen."

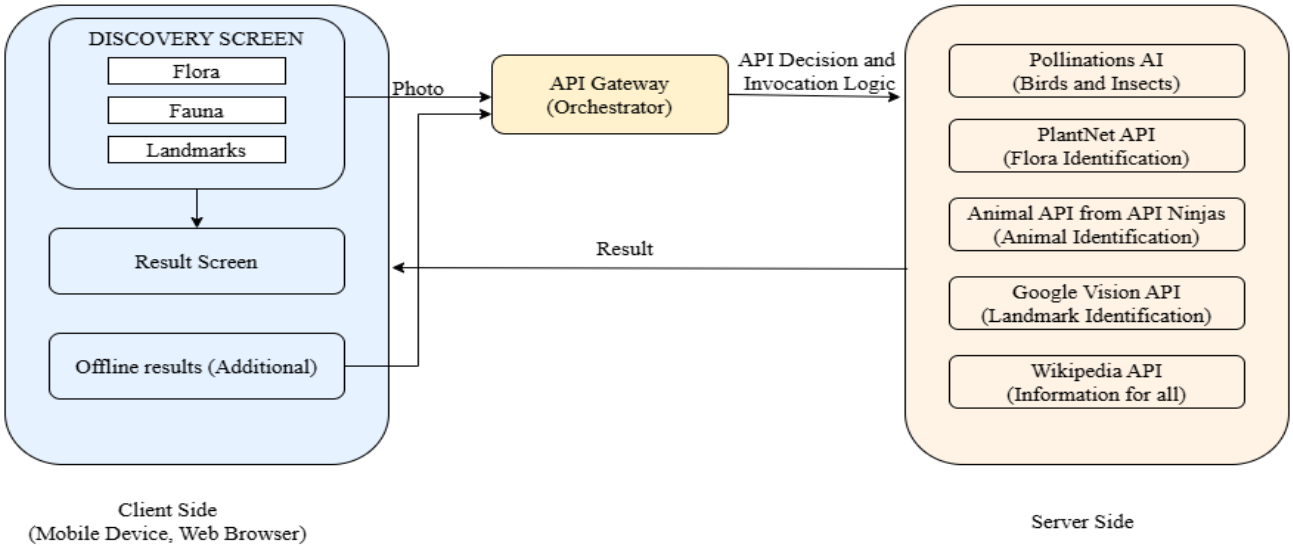

Figure 3 : System Architecture

4.3 UML DIAGRAMS

4.3.1 USE CASE DIAGRAM

An image discovery application's microservices-based architecture is shown in this class diagram. The DiscoveryScreen, which offers user interaction points like selectFlora(), selectFauna(), selectLandmark(), and displayResults(), is the main component. This screen is directly interacted with by a user.

The APIGateway serves as an orchestrator for different image processing tasks, receiving image return results from the DiscoveryScreen. The APIGateway provides methods that accept images as input, such as processFlora(), processFauna(), and processLandmark().

The APIGateway interfaces with PlantNetAPI (identifyPlant) for Flora, AnimalAPI (identifyAnimal) for Fauna, and GoogleVisionAPI (identifyLandmark) for Landmark. Importantly, in order to enrich the discovery results before they are sent back to the user, the APIGateway also communicates with WikipediaAPI to obtainDetails based on a keyword.

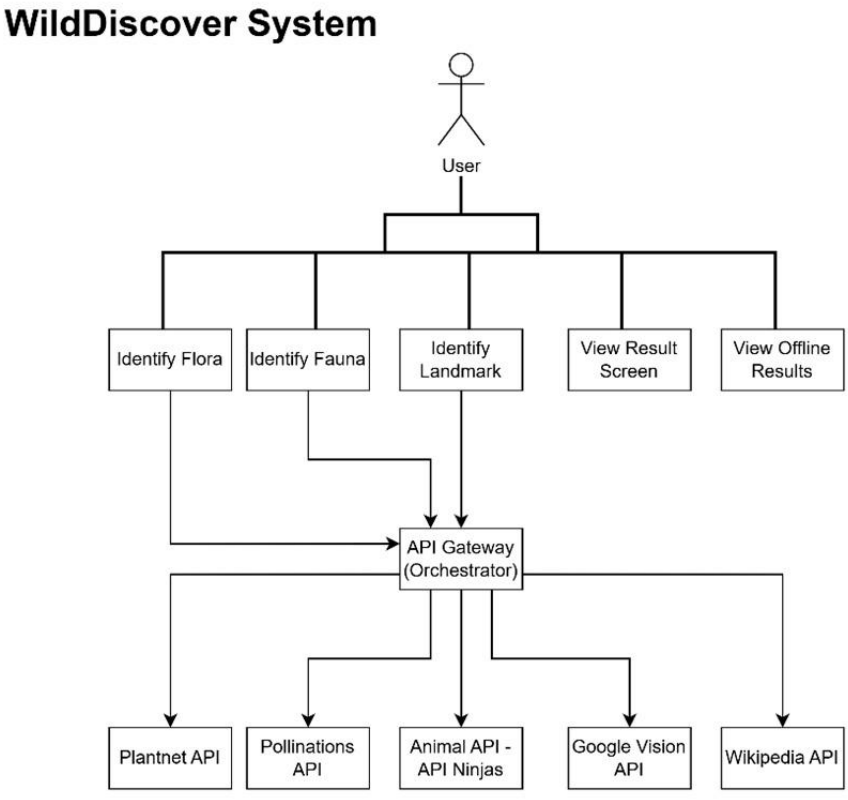

Figure 4 : Use Case Diagram

4.3.2 SEQUENCE DIAGRAM

The sequence diagram depicts the processes involved and the sequence of messages exchange between the processes needed to carry out the functionalities.

The sequence of the application is as follows:

1. User Starts Discovery: On the Discovery Screen, the user chooses either the Fauna, Flora, or Landmark discovery types and uploads an image.
2. API Gateway Orchestration: The image is sent to the API Gateway from the Discovery Screen, and it serves as an orchestrator for the ensuing API calls.
3. Specialized API Identification: To identify the subject, the API Gateway sends the image to the appropriate specialized API (e.g., Google Vision for Landmark, Animal API for Fauna, or PlantNet for Flora).
4. Detailed Information Retrieval: The API Gateway asks the Wikipedia API for more specific information about the identified subject after obtaining initial identification.
5. Results Display: The Discovery Screen presents the user with the final data after receiving the compiled results from the API Gateway.

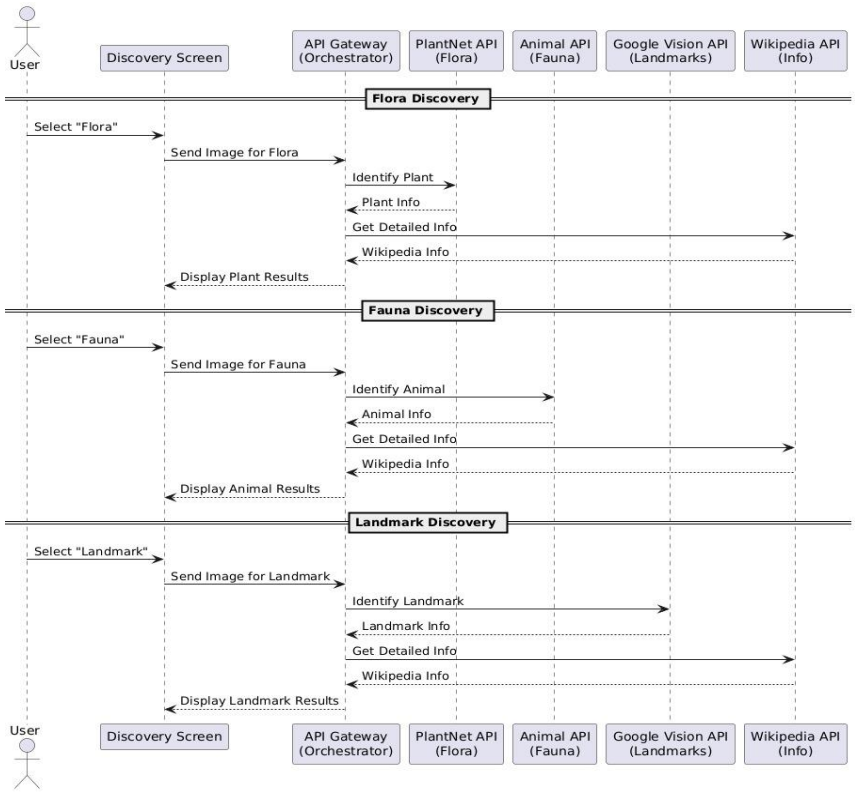

Figure 5 : Sequence Diagram

4.3.3 ACTIVITY DIAGRAM

This class diagram depicts a microservices-based architecture for an image discovery application. The central component is the DiscoveryScreen, which provides user interaction points like selectFlora(), selectFauna(), selectLandmark(), and displayResults(). A User directly interacts with this screen.

The DiscoveryScreen sends image returns results to the APIGateway, which acts as an orchestrator for various image processing tasks. The APIGateway exposes methods like processFlora(), processFauna(), and processLandmark(), each taking an Image as input.

The APIGateway for Flora communicates with PlantNetAPI (identifyPlant), for Fauna with AnimalAPI (identifyAnimal), and for Landmark with GoogleVisionAPI (identifyLandmark). Crucially, for extra info, the APIGateway also interacts with WikipediaAPI to getDetails based on a keyword, enriching the discovery results before they are returned to the user.

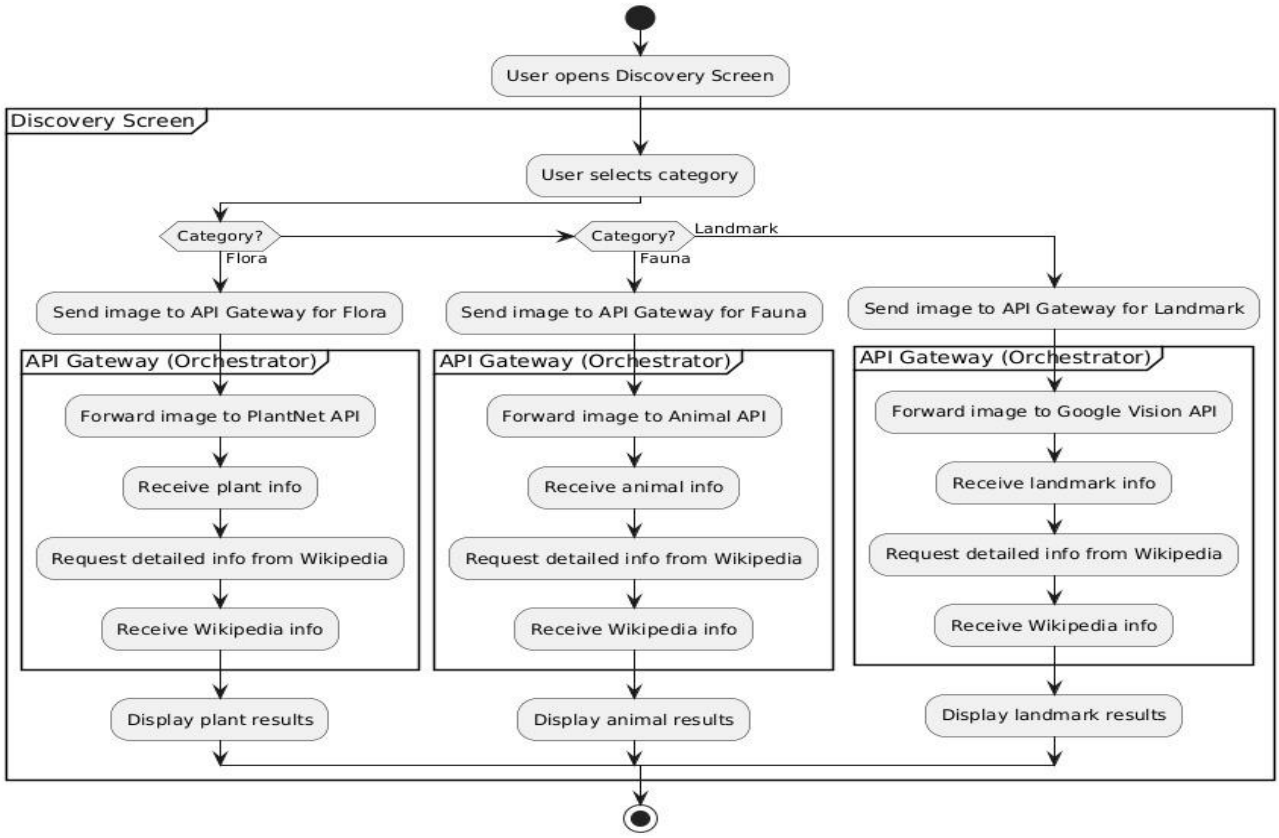

Figure 6 : Activity Diagram

ML MODEL

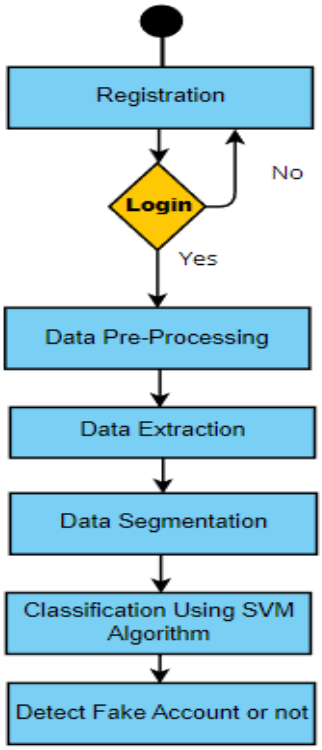

Figure 7 : ML Model

1. The initial state of the project includes providing account details as input by the user to the developed web application.
2. The input account details are passed to the model. The model is developed using a classification algorithm called Support Vector Machine.
3. The classifier does classification of the account and provides a class label to the input account.
4. The final output depends on the class label of the account which is as follows:
    - If the class label of the account is fraudulent then the output will be given as “FAKE”.
    - If the class label is not fraudulent then the output will be given as “Not Fake”.

SIMILARITY INDEX - PROFILE CLONING

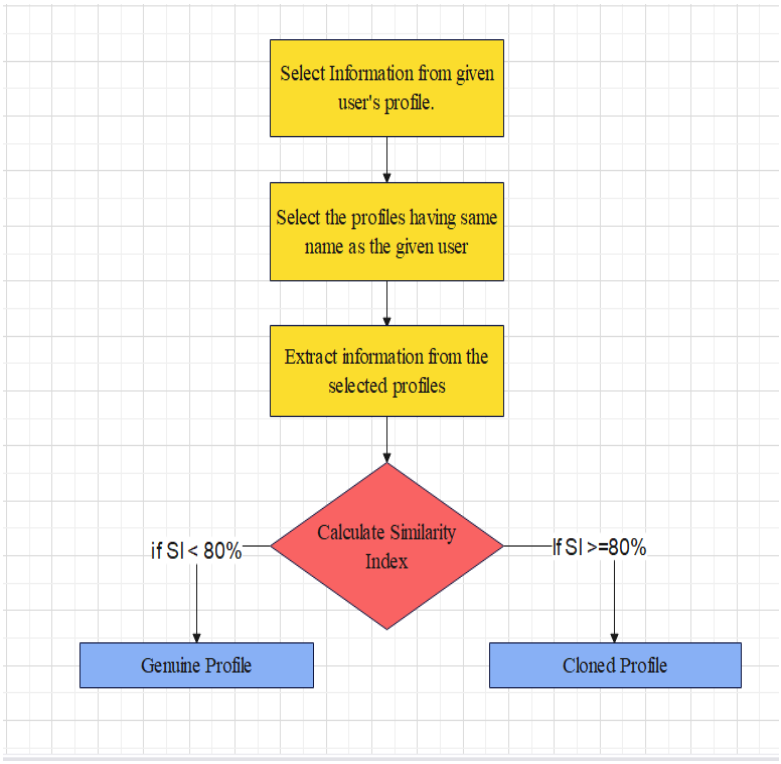

Figure 8 : Similarity Index

1. The initial state of the project includes providing input account details by the user to the developed web application.
2. The classifier does classification of accounts and provides a class label to the input account details.
3. The final output depends on the class label of the account which is as follows:
   - If the class label of the account is Fraudulent then the output will be given as "Fake Account".
   - If the class label of the account is not Fraudulent then the output will be given as "Real Account".

4.4 MODULE DIAGRAM

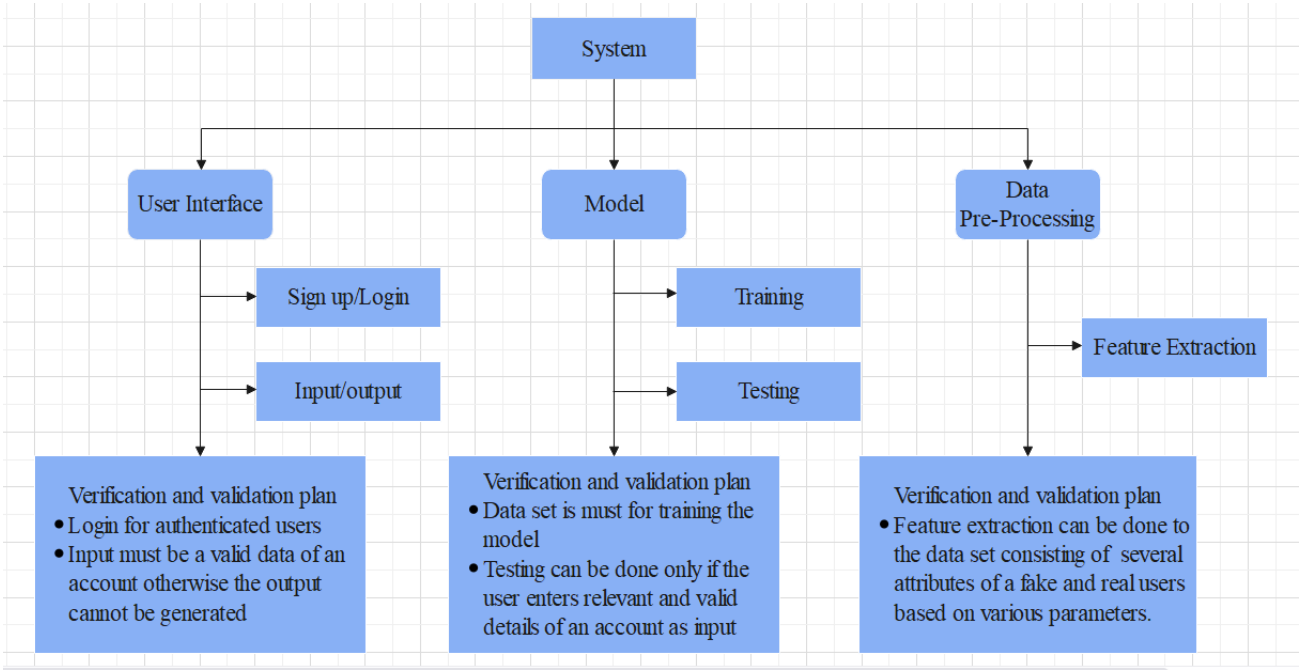

Figure 9 : Module Diagram

The proposed system consists of three modules as mentioned in the above diagram.

- User Interface
- ML Model
- Data Pre-Processing

USER INTERFACE

The functions in the user interface module includes users registering/signing before doing any operations. Once the user registers, their details will be stored in the database. This module allows only the authenticated users to login and output can only be generated if the input data or details entered by the user is valid. The output determines the class of an account.

DATA PRE-PROCESSING

Data normalization technique is used on the dataset, which removes the sophisticated noise, before analyzing them. Feature extraction is the major functionality of the data pre-processing module where various features are extracted from the dataset consisting of attributes of fake and real accounts that is used to train the model.

ML MODEL

The model uses the dataset after feature extraction for the purpose of training. After the model is trained, the input data from the user is taken as test data to determine the class of the account.

CHAPTER 5

IMPLEMENTATION

5.1 Source Code

App.js

```
import React, { useEffect } from 'react';
import { NavigationContainer, DefaultTheme, DarkTheme } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { useColorScheme, Alert } from 'react-native';
import * as ImagePicker from 'expo-image-picker';
import { AppModeProvider, useAppMode } from './context/AppModeContext';
import HomeScreen from './screens/HomeScreen';
import FloraScreen from './screens/FloraScreen';
import FaunaScreen from './screens/FaunaScreen';
import LandmarkScreen from './screens/LandmarkScreen';
import ResultScreen from './screens/ResultScreen';
import FloraScreenOffline from './screens/FloraScreenOffline';
import FaunaScreenOffline from './screens/FaunaScreenOffline';
import LandmarkScreenOffline from './screens/LandmarkScreenOffline';
const Stack = createStackNavigator();
function AppNavigator() {
 const theme = useColorScheme();
 const { isOffline } = useAppMode();
 return (
  <NavigationContainer theme={theme === 'dark' ? DarkTheme : DefaultTheme}>
   <Stack.Navigator initialRouteName="Home">
    <Stack.Screen name="Home" component={HomeScreen} options={{ headerShown: false }} />
    <Stack.Screen
     name="FloraScreen"
     component={isOffline ? FloraScreenOffline : FloraScreen}
     options={{ title: 'Identify Flora' }}
    />
    <Stack.Screen
     name="FaunaScreen"
     component={isOffline ? FaunaScreenOffline : FaunaScreen}
```

```
          options={{ title: 'Identify Fauna' }}
        />
        <Stack.Screen
          name="LandmarkScreen"
          component={isOffline ? LandmarkScreenOffline : LandmarkScreen}
          options={{ title: 'Identify Landmark' }}
        />
        <Stack.Screen name="ResultScreen" component={ResultScreen} options={{ title: 'Identification Result' }} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
export default function App() {
  useEffect(() => {
    (async () => {
      try {
        const { status: cameraStatus } = await ImagePicker.requestCameraPermissionsAsync();
        const { status: mediaStatus } = await ImagePicker.requestMediaLibraryPermissionsAsync();
        if (cameraStatus !== 'granted' || mediaStatus !== 'granted') {
          Alert.alert('Permissions Required', 'Camera and media permissions are required to use this app.');
        }
      } catch (error) {
        console.error('Permission request error:', error);
      }
    })();
  }, []);
  return (
    <AppModeProvider>
      <AppNavigator />
    </AppModeProvider>
  );
}
```

HomeScreen.js

```
import React, { useEffect, useCallback, useState } from "react";
import {
```

```
  View,
  Text,
  TouchableOpacity,
  useColorScheme,
  Modal,
  TextInput,
  Alert,
  Button,
} from "react-native";
import { useNavigation } from "@react-navigation/native";
import { FontAwesome5, Ionicons } from "@expo/vector-icons";
import { StatusBar } from "expo-status-bar";
import { useFonts, Poppins_700Bold, Poppins_400Regular } from "@expo-google-fonts/poppins";
import * as SplashScreen from "expo-splash-screen";
import { useAppMode } from "../context/AppModeContext";
import * as SecureStore from "expo-secure-store";
import History from "./History";
SplashScreen.preventAutoHideAsync();
const categories = [
  { name: "Flora", icon: "seedling", color: "#EAF6E9", route: "FloraScreen" },
  { name: "Fauna", icon: "dove", color: "#E9F2FF", route: "FaunaScreen" },
  { name: "Landmark", icon: "landmark", color: "#FFF7E6", route: "LandmarkScreen" },
];
export default function HomeScreen() {
  const [showHistory, setShowHistory] = useState(false);
  const [phone, setPhone] = useState("");
  const [isPhoneModalVisible, setPhoneModalVisible] = useState(false);
  const navigation = useNavigation();
  const theme = useColorScheme();
  const { isOffline, setIsOffline } = useAppMode();
  const [fontsLoaded] = useFonts({
    Poppins_700Bold,
    Poppins_400Regular,
  });
  useEffect(() => {
```

```
  const checkPhone = async () => {
    const savedPhone = await SecureStore.getItemAsync("phone");
    if (!savedPhone) {
      setPhoneModalVisible(true);
    } else {
      setPhone(savedPhone);
    }
  };
  checkPhone();
}, []);
const isValidPhone = (num) => /^[0-9]{10}$/.test(num);
const handleSavePhone = async () => {
  if (isValidPhone(phone)) {
    await SecureStore.setItemAsync("phone", phone);
    setPhoneModalVisible(false);
  } else {
    Alert.alert("Invalid Number", "Please enter a valid 10-digit phone number.");
  }
};
const onLayoutRootView = useCallback(async () => {
  if (fontsLoaded) {
    await SplashScreen.hideAsync();
  }
}, [fontsLoaded]);
if (!fontsLoaded) {
  return null;
}
return (
  <View onLayout={onLayoutRootView}>
    <StatusBar style={theme === "dark" ? "light" : "dark"} />
    <View>
      <Text> 🌿 WildDiscover □</Text>

      <TouchableOpacity
        onPress={() => {
```

```
      setIsOffline((prev) => {
        const newState = !prev;
        Alert.alert(
          "Mode Switched",
          newState ? "🛰️ You are now in Offline Mode." : "🌐 Switched back to Online Mode.",
          [{ text: "OK" }]
        );
        return newState;
      });
    }}
  >
    <Ionicons
      name={isOffline ? "cloud-offline" : "cloud-outline"}
      size={24}
      color={theme === "dark" ? "white" : "black"}
    />
  </TouchableOpacity>
</View>
<Text>
  Identify and explore wildlife & landmarks around you!
</Text>
<View>
  {categories.map((category) => (
    <TouchableOpacity
      key={category.name}
      onPress={() =>
        navigation.navigate(isOffline ? `${category.name}ScreenOffline` : category.route)
      }
      activeOpacity={0.7}
    >
      <FontAwesome5 name={category.icon} size={40} color="black" />
      <Text>{category.name}</Text>
    </TouchableOpacity>
  ))}
  <TouchableOpacity
```

```
        onPress={() => setShowHistory(true)}
        activeOpacity={0.7}
      >
        <FontAwesome5 name="history" size={40} color="black" />
        <Text>History</Text>
      </TouchableOpacity>
    </View>
    <Text>
      📸 Capture or Upload to Identify
    </Text>
    {/* Phone Number Modal */}
    <Modal visible={isPhoneModalVisible} transparent animationType="slide">
      <View>
        <View>
          <Text>Enter Your Phone Number</Text>
          <TextInput
            placeholder="e.g. 9876543210"
            keyboardType="phone-pad"
            value={phone}
            onChangeText={setPhone}
            maxLength={10}
          />
          <Button title="Save" onPress={handleSavePhone} />
        </View>
      </View>
    </Modal>
    {/* History Modal */}
    {showHistory && <History visible={showHistory} onClose={() => setShowHistory(false)} />}
  </View>
 );
}
```

FaunaScreen.js

```
import React, { useState } from "react";
import { View, Text, TouchableOpacity, Image, useColorScheme, Alert } from "react-native";
import * as ImagePicker from "expo-image-picker";
```

```
import { Ionicons } from "@expo/vector-icons";
import { StatusBar } from "expo-status-bar";
import { useNavigation } from "@react-navigation/native";
import { detectObject } from "../api/detectionAPI.js"; // Import the detection API
import { validateLabelWithGemini } from "../api/geminiAPI.js"; // Import Gemini validation API
export default function FaunaScreen() {
  const [selectedImage, setSelectedImage] = useState(null);
  const theme = useColorScheme(); // Detect system theme
  const navigation = useNavigation(); // Get navigation object
  const pickImage = async () => {
    let result = await ImagePicker.launchImageLibraryAsync({
      mediaTypes: ImagePicker.MediaTypeOptions.Images,
      allowsEditing: true,
      aspect: [4, 3],
      quality: 1,
    });
    if (!result.canceled && result.assets && result.assets.length > 0) {
      setSelectedImage(result.assets[0].uri);
    } else {
      Alert.alert("Image Selection Failed", "Please select a valid image.");
    }
  };
  const takePhoto = async () => {
    let result = await ImagePicker.launchCameraAsync({
      allowsEditing: true,
      aspect: [4, 3],
      quality: 1,
    });
    if (!result.canceled && result.assets && result.assets.length > 0) {
      setSelectedImage(result.assets[0].uri);
    } else {
      Alert.alert("Image Selection Failed", "Please select a valid image.");
    }
  };
  const analyzeImage = async () => {
```

```
 if (!selectedImage) {
   Alert.alert("No Image Selected", "Please choose an image first!");
   return;
 }
 try {
   const result = await detectObject(selectedImage, "fauna");
   const isValid = await validateLabelWithGemini(result.description, "fauna");
   if (isValid) {
 navigation.navigate("ResultScreen", {
   result: { ...result,
 name: result.description?.slice(0, 20) || '',
 imageUri: selectedImage
 },
   type: "fauna"
 });
} else {
 navigation.navigate("ResultScreen", {
   result: { tag: "Unrecognized", name: "Unrecognized", imageUri: selectedImage },
   type: "fauna"
 });
}
 } catch (error) {
   Alert.alert("Error", "Failed to analyze image. Please try again.");
 }
};
 return (
   <View>
     <StatusBar style={theme === "dark" ? "light" : "dark"} />

     <Text> 🦉 Identify Fauna</Text>
     <Text>
       Capture or upload an image to identify animals, birds, or insects around you.
     </Text>
     {selectedImage ? (
       <Image source={{ uri: selectedImage }} />
```

```
    ) : (
      <View>
        <Ionicons name="paw-outline" size={100} color="#FF9800" />
        <Text>No Image Selected</Text>
      </View>
    )}
    <View>
      <TouchableOpacity onPress={takePhoto}>
        <Ionicons name="camera" size={22} color="white" />
        <Text>Take Photo</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={pickImage}>
        <Ionicons name="image" size={22} color="white" />
        <Text>Upload Image</Text>
      </TouchableOpacity>
    </View>
    <TouchableOpacity
      onPress={analyzeImage}
      disabled={!selectedImage}
    >
      <Text>{selectedImage ? "Analyze Image" : "Select Image First"}</Text>
    </TouchableOpacity>
  </View>
 );
}
```

FloraScreen.js

```
import React, { useState } from "react";
import { View, Text, TouchableOpacity, Image, useColorScheme, Alert } from "react-native";
import * as ImagePicker from "expo-image-picker";
import { Ionicons } from "@expo/vector-icons";
import { StatusBar } from "expo-status-bar";
import { useNavigation } from "@react-navigation/native";
import { detectObject } from "../api/detectionAPI.js"; // Import the detection API
import { validateLabelWithGemini } from "../api/geminiAPI.js";
export default function FloraScreen() {
```

```
const [selectedImage, setSelectedImage] = useState(null);
const theme = useColorScheme(); // Detect system theme
const navigation = useNavigation(); // Get navigation object
const pickImage = async () => {
  let result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Images,
    allowsEditing: true,
    aspect: [4, 3],
    quality: 1,
  });
  if (!result.canceled) {
    setSelectedImage(result.assets[0].uri);
  }
};
const takePhoto = async () => {
  let result = await ImagePicker.launchCameraAsync({
    allowsEditing: true,
    aspect: [4, 3],
    quality: 1,
  });
  if (!result.canceled) {
    setSelectedImage(result.assets[0].uri);
  }
};
const analyzeImage = async () => {
  if (!selectedImage) {
    Alert.alert("No Image Selected", "Please choose an image first!");
    return;
  }
  try {
    const result = await detectObject(selectedImage, "flora");
    const isValid = await validateLabelWithGemini(result.description, "flora");
    if (isValid) {
  navigation.navigate("ResultScreen", {
    result: { ...result,
```

```
 name: result.description?.slice(0, 20) || '',
 imageUri: selectedImage
},
    type: "fauna"
  });
 } else {
  navigation.navigate("ResultScreen", {
    result: { tag: "Unrecognized", name: "Unrecognized", imageUri: selectedImage },
    type: "fauna"
  });
 }
  } catch (error) {
    console.log("Error analyzing image:", error);
    Alert.alert("Error", "Failed to analyze image. Please try again.");
  }
 };
 return (
  <View>
    <StatusBar style={theme === "dark" ? "light" : "dark"} />
    <Text>🌱 Identify Flora</Text>
    <Text>
      Capture or upload an image to identify plants around you.
    </Text>
    {selectedImage ? (
      <Image source={{ uri: selectedImage }} />
    ) : (
      <View>
        <Ionicons name="leaf-outline" size={100} color="#4CAF50" />
        <Text>
          No Image Selected
        </Text>
      </View>
    )}
    <View>
      <TouchableOpacity onPress={takePhoto}>
```

```
        <Ionicons name="camera" size={22} color="white" />
        <Text>Take Photo</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={pickImage}>
        <Ionicons name="image" size={22} color="white" />
        <Text>Upload Image</Text>
      </TouchableOpacity>
    </View>
    <TouchableOpacity
      onPress={analyzeImage}
      disabled={!selectedImage}
    >
      <Text>{selectedImage ? "Analyze Image" : "Select Image First"}</Text>
    </TouchableOpacity>
  </View>
 );
}
```

LandmarkScreen.js

```
import React, { useState } from "react";
import { View, Text, TouchableOpacity, Image, useColorScheme, Alert } from "react-native";
import * as ImagePicker from "expo-image-picker";
import { Ionicons } from "@expo/vector-icons";
import { StatusBar } from "expo-status-bar";
import { useNavigation } from "@react-navigation/native";
import { detectObject } from "../api/detectionAPI.js";
import { validateLabelWithGemini } from "../api/geminiAPI.js";
export default function LandmarkScreen() {
  const [selectedImage, setSelectedImage] = useState(null);
  const theme = useColorScheme(); // Detect system theme
  const navigation = useNavigation();
const pickImage = async () => {
   let result = await ImagePicker.launchImageLibraryAsync({
     mediaTypes: ImagePicker.MediaTypeOptions.Images,
     allowsEditing: true,
     aspect: [4, 3],
```

```
      quality: 1,
    });
    if (!result.canceled && result.assets?.length > 0) { // Fix property name
      setSelectedImage(result.assets[0].uri);
    } else {
      Alert.alert("Image Selection Failed", "Please select a valid image.");
    }
  };
const takePhoto = async () => {
    let result = await ImagePicker.launchCameraAsync({
      allowsEditing: true,
      aspect: [4, 3],
      quality: 1,
    });
    if (!result.canceled && result.assets?.length > 0) { // Fix property name
      setSelectedImage(result.assets[0].uri);
    } else {
      Alert.alert("Image Selection Failed", "Please select a valid image.");
    }
  };
const analyzeImage = async () => {
    if (!selectedImage) {
      Alert.alert("No Image Selected", "Please choose an image first!");
      return;
    }
    try {
      const result = await detectObject(selectedImage, "landmark");
      const isValid = await validateLabelWithGemini(result.description, "landmark");
      if (isValid) {
    navigation.navigate("ResultScreen", {
      result: { ...result,
  name: result.description?.slice(0, 20) || '',
  imageUri: selectedImage
  },
      type: "fauna"
```

```
  });
} else {
  navigation.navigate("ResultScreen", {
    result: { tag: "Unrecognized", name: "Unrecognized", imageUri: selectedImage },
    type: "fauna"
  });
}
  } catch (error) {
    Alert.alert("Error", "Failed to analyze image. Please try again.");
  }
};
return (
  <View>
    <StatusBar style={theme === "dark" ? "light" : "dark"} />
    <Text>🏛️ Identify Landmarks</Text>
    <Text>
      Capture or upload an image to identify famous landmarks.
    </Text>
    {selectedImage ? (
      <Image source={{ uri: selectedImage }} />
    ) : (
      <View>
        <Ionicons name="business-outline" size={100} color="#4A90E2" />
        <Text>
          No Image Selected
        </Text>
      </View>
    )}
    <View>
      <TouchableOpacity onPress={takePhoto}>
        <Ionicons name="camera" size={22} color="white" />
        <Text>Take Photo</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={pickImage}>
        <Ionicons name="image" size={22} color="white" />
```

```
        <Text>Upload Image</Text>
      </TouchableOpacity>
    </View>
    <TouchableOpacity
      onPress={analyzeImage}
      disabled={!selectedImage}
    >
      <Text>{selectedImage ? "Analyze Image" : "Select Image First"}</Text>
    </TouchableOpacity>
  </View>
 );
}
```

ResultScreen.js

```
import React from 'react';
import { View, Text, Image, useColorScheme, Linking, ScrollView } from 'react-native';
import { StatusBar } from 'expo-status-bar';
import * as SecureStore from 'expo-secure-store';
import * as FileSystem from 'expo-file-system';
import { Platform } from 'react-native';
import { useEffect } from 'react';
export default function ResultScreen({ route }) {
 useEffect(() => {
 const uploadToBackend = async () => {
   try {
    const phone = await SecureStore.getItemAsync("phone");
    if (!phone || !result?.imageUri || !result?.name) {
      console.warn("Missing phone, image or tag");
      return;
    }
    const uriParts = result.imageUri.split(".");
    const fileType = uriParts[uriParts.length - 1];
    const file = {
      uri: result.imageUri,
      name: `image.${fileType}`,
      type: `image/${fileType}`,
```

```
      };
      const formData = new FormData();
      formData.append('phone', phone);
      formData.append('tag', result.name);
      formData.append('image', file);
      const res = await fetch("http://192.168.137.222:3000/upload", {
        method: "POST",
        body: formData,
        headers: {
          'Content-Type': 'multipart/form-data',
        },
      });
      const responseJson = await res.json();
      if (responseJson.success) {
        console.log("Upload successful:", responseJson.imageUrl);
      } else {
        console.warn("Upload failed:", responseJson.error);
      }
    } catch (err) {
      console.error("Error uploading to backend:", err);
    }
  };
  uploadToBackend();
}, []);
  const { result, type } = route.params;
  const theme = useColorScheme(); // Detect system theme
console.log("ResultScreen Data:", result);
  const openInGoogleMaps = (latitude, longitude) => {
    const url = `https://www.google.com/maps/search/?api=1&query=${latitude},${longitude}`;
    Linking.openURL(url);
  };
  const shouldDisplayDetail = (detail) => {
    return detail && !["Unknown", "N/A", "Not Available"].includes(detail);
  };
  return (
```

```
    <ScrollView>
      <StatusBar style={theme === "dark" ? "light" : "dark"} />
      <Text>🔍 Identification Result</Text>
      {/* Centered Image */}
      <View>
        {result.imageUri && <Image source={{ uri: result.imageUri }} />}
      </View>
      {/* Details Section */}
      <View>
        {/* Name (Bold & Bigger) */}
        {shouldDisplayDetail(result.name) && (
          <Text>
            Name: {result.name}
          </Text>
        )}
        {shouldDisplayDetail(result.scientificName) && (
          <Text>
            Scientific Name: {result.scientificName} {result.scientificNameAuthorship ?
`(${result.scientificNameAuthorship})` : ""}
          </Text>
        )}
        {/* 📝 Description (Only Display if Available) */}
        {shouldDisplayDetail(result.description) && (
          <Text>
            📝 Description: {result.description}
          </Text>
        )}
        {/* 🌿 Flora Section */}
        {type === "flora" && (
          <View>
            <Text>🌿 Flora Details</Text>
            {shouldDisplayDetail(result.commonNames) && result.commonNames.length > 0 && (
              <Text>📗 Common Names: {result.commonNames.join(", ")}</Text>
            )}
```

```
        {shouldDisplayDetail(result.genus) && (
          <Text>🌱 Genus: {result.genus} {result.genusAuthorship ? `(${result.genusAuthorship})` : ""}</Text>
        )}
        {shouldDisplayDetail(result.family) && (
          <Text>🌳 Family: {result.family} {result.familyAuthorship ? `(${result.familyAuthorship})` : ""}</Text>
        )}
        {shouldDisplayDetail(result.habitat) && <Text>🏡 Habitat: {result.habitat}</Text>}
        {shouldDisplayDetail(result.plantType) && <Text>🌾 Plant Type: {result.plantType}</Text>}
        {shouldDisplayDetail(result.floweringSeason) && <Text>🌸 Flowering Season:
{result.floweringSeason}</Text>}
        {shouldDisplayDetail(result.uses) && <Text>💡 Uses: {result.uses}</Text>}
        {result.remainingIdentificationRequests !== undefined && (
          <Text>🔢 Remaining API Requests: {result.remainingIdentificationRequests}</Text>
        )}
      </View>
    )}
    {/* 🦁 Fauna Section */}
    {type === "fauna" && (
      <View>
        <Text>🦁 Fauna Details</Text>
        {shouldDisplayDetail(result.scientificName) && <Text>🔬 Scientific Name: {result.scientificName}</Text>}
        {shouldDisplayDetail(result.family) && <Text>🌳 Family: {result.family}</Text>}
        {shouldDisplayDetail(result.diet) && <Text>🍽️ Diet: {result.diet}</Text>}
        {shouldDisplayDetail(result.lifespan) && <Text>⏳ Lifespan: {result.lifespan}</Text>}
        {shouldDisplayDetail(result.habitat) && <Text>🏡 Habitat: {result.habitat}</Text>}
        {shouldDisplayDetail(result.conservationStatus) && <Text>🌍 Conservation Status:
{result.conservationStatus}</Text>}
        {shouldDisplayDetail(result.prey) && <Text>🦌 Prey: {result.prey}</Text>}
        {shouldDisplayDetail(result.groupBehavior) && <Text>👥 Group Behavior: {result.groupBehavior}</Text>}
        {shouldDisplayDetail(result.topSpeed) && <Text>🏃 Top Speed: {result.topSpeed}</Text>}
        {shouldDisplayDetail(result.weight) && <Text>⚖️ Weight: {result.weight}</Text>}
        {shouldDisplayDetail(result.height) && <Text>📏 Height: {result.height}</Text>}
        {result.name === "Needs Verification" && (
```

```
          <Text>
            ⚠️ This identification has low confidence. Please verify the result.
          </Text>
        )}
      </View>
    )}
    {/* 📍 Landmark Section */}
    {type === "landmark" && (
      <View>
        <Text>📍 Landmark Details</Text>
        {shouldDisplayDetail(result.name) && <Text>🏛️ Name: {result.name}</Text>}
        {shouldDisplayDetail(result.location) && typeof result.location === 'string' && (
          <Text>📍 Location: {result.location}</Text>
        )}
        {result.latitude != null && result.longitude != null && (
          <Text onPress={() => openInGoogleMaps(result.latitude, result.longitude)}>
            🌍 Coordinates: {parseFloat(result.latitude).toFixed(4)}, {parseFloat(result.longitude).toFixed(4)}
          </Text>
        )}
        {shouldDisplayDetail(result.built) && <Text>🏗️ Built: {result.built}</Text>}
        {shouldDisplayDetail(result.architect) && <Text>👨‍🎨 Architect: {result.architect}</Text>}
        {shouldDisplayDetail(result.style) && <Text>🏛️ Architectural Style: {result.style}</Text>}
        {shouldDisplayDetail(result.significance) && <Text>📜 Significance: {result.significance}</Text>}
      </View>
    )}
    {/* Wikipedia Link */}
    {result.wikipediaLink && result.wikipediaLink.startsWith("http") && (
      <Text onPress={() => Linking.openURL(result.wikipediaLink)}>
        Learn more on Wikipedia
      </Text>
    )}
  </View>
</ScrollView>
);}
```

CHAPTER 6

RESULTS

6.1 RESULT

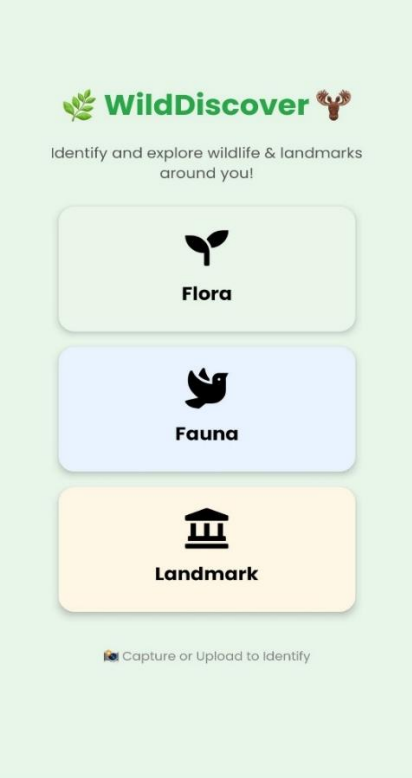

Figure 10: User Interface

Figure 11: Flora Result Screen

Figure 12: Fauna Result Screen

Figure 13: Landmark Result Screen

6.2 VERIFICATION AND VALIDATION

Verification

Verification ensures that the system is being built correctly — that is, it meets the technical specifications and design requirements. In the context of WildDiscover, verification involves checking if the mobile application adheres to the defined functional and non-functional requirements.

Techniques Used:

Code Reviews: Regular peer reviews were conducted to identify bugs and improve logic and maintainability.

Static Testing: Linting tools were used to analyze React Native and backend code without executing it.

Unit Testing: Key modules such as image upload, API integration (e.g., PlantNet, Google Vision), and offline ML predictions were tested independently.

Offline Mode Verification: Verified that the application can operate without network connectivity by using on-device MobileNet predictions for flora, fauna, and landmark classification.

Verification Checklist:

Table 1: Verification Checklist

Component	**Remarks**
User Interface	Functional and responsive
Image Upload/Camera	Supports both modes
API Integration	Google Vision, PlantNet, Pollinations
On-device ML Inference	MobileNet used for offline predictions
Wikipedia Integration	Data retrieved and displayed successfully
AWS S3 Upload	Verified with successful image storage
Mode Switching (Online/Offline)	Verified toggle behavior and flow

Validation

Validation ensures that the right system is being built — it checks if the system meets user needs and expectations.

Validation Approach:

- **Functional Testing:** End-to-end testing was performed to ensure correct behavior when users upload or capture images for identification.
- **Usability Testing:** Conducted informal user testing with students and faculty to assess the UI, ease of navigation, and result interpretation.
- **Performance Testing:** Measured response time of image recognition APIs and offline model inference to ensure timely feedback.
- **Data Accuracy Testing:** Compared the classification results from the model/API against known species and landmarks to validate correctness.

Validation Metrics:

Table 2: Validation Metrics

Metric	Target	Achieved
Image classification accuracy	≥ 85%	~88% (on curated dataset)
Identification time	< 5 seconds per image	2–4 seconds (API), <1 sec (offline)
User satisfaction (feedback)	≥ 4/5 rating	4.3 average (from 10+ test users)

CHAPTER 7

CONCLUSION

CONCLUSION

The increasing demand for intelligent and easily accessible nature exploration tools is met by WildDiscover. The application assists users in correctly identifying plants, animals, and landmarks by combining on-device machine learning with image recognition APIs. The system guarantees a smooth and educational user experience by providing offline functionality, cloud storage, and Wikipedia integration. Users can confidently and easily explore and learn about their surroundings with the help of the suggested application.

FUTURE ENHANCEMENT

- **User History and Personal Logs** – Allow users to view and manage past identifications with metadata.

- **Geolocation-Based Suggestions** – Recommend species or landmarks based on the user's current location.

- **Multilingual Support** – Add regional language options for broader accessibility.

- **Improved Personalization with AI** – Offer content and identification suggestions based on user preferences and behaviour.

- **Smart Caching and Rate Limit Handling** – Optimize API usage by storing frequent results locally and managing request limits.

REFERENCES

1. S. B. Neupane, K. Sato, and B. P. Gautam, "Computer Vision Techniques in Wildlife Monitoring," *Journal of International Scientific Publications: Materials, Methods & Technologies*, 2022. https://www.scientific-publications.net
2. D. Mpouziotas, P. Karvelis, I. Tsoulos, and C. Stylios, "Automated Wildlife Bird Detection from Drone Footage Using Computer Vision Techniques," *Applied Sciences*, vol. 13, no. 13, Article 7787, 2023. https://www.mdpi.com/2076-3417/13/13/7787
3. S. B. Islam and D. Valles, "Animal Species Recognition with Deep Convolutional Neural Networks from Ecological Camera Trap Images," *Animals*, vol. 13, no. 9, Article 1526, 2023. https://www.mdpi.com/2076-2615/13/9/1526
4. J. Abdollahi, "Identification of Medicinal Plants in Ardabil Using DL," *IEEE Conference Publication*, 2022. https://ieeexplore.ieee.org
5. A. Hernandez, Z. Miao, L. Vargas, S. Beery, and R. Dodhia, "Pytorch-Wildlife: A Collaborative Deep Learning Framework for Conservation," *arXiv preprint*, arXiv:2405.12930, 2024. https://arxiv.org/abs/2405.12930
6. M. E. Hossain, M. A. Kabir, L. Zheng, and D. L. Swain, "Deep Learning for Wildlife Conservation and Habitat Monitoring: A Comprehensive Review," *ScienceDirect*, 2022. https://www.sciencedirect.com
7. C. Jo, D. Hwang, S. Ko, M. H. Yang, and M. C. Lee, "Deep Learning-Based Landmark Recognition and Angle Measurement of Full-Leg Radiographs," *Knee Surgery, Sports Traumatology, Arthroscopy*, 2022. https://link.springer.com

8. M. N. Razali and E. O. N. Tony, "Landmark Recognition Model for Smart Tourism Using Lightweight Deep Learning and Linear Discriminant Analysis," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 14, no. 2, 2023. https://thesai.org
9. J. Deka, S. Laskar, and B. Baklial, "Automated Freshwater Fish Species Classification Using Deep CNN," *Journal of The Institution of Engineers (India): Series B*, 2023. https://link.springer.com
10. S. Ghosh, A. Singh, K. Kavita, N. Z. Jhanjhi, M. Masud, and S. Aljahdali, "SVM and KNN Based CNN Architectures for Plant Classification," *Computers, Materials & Continuation*, 2022. https://www.techscience.com/cmc
11. K. Takaya, Y. Taguchi, and T. Ise, "Individual Identification of Japanese Giant Salamanders Using Deep Learning," *Public Library of Medicine (PMC)*, 2024. https://www.ncbi.nlm.nih.gov/pmc/
12. A. Carvajal and V. R. Garcia-Colon, "Detection of Wildlife Animals Using Deep Learning Approaches: A Systematic Review," *IEEE Xplore*, 2022. https://ieeexplore.ieee.org

GitHub Link :

http://Github.com/V-Satwik-Reddy/WildDiscover

https://github.com/V-Satwik-Reddy/WildDiscoverB